# INTEGER BASIC*+

## A RELOCATABLE RAM VERSION OF APPLE ][’s FIRST LANGUAGE INTEGER BASIC*

Developed By

**JIM McBRIDE**

and

**GARTH HITCHENS**



## OVERVIEW

Integer Basic Plus was developed to fill a need for those people who bought an Apple II Plus without the Integer Firmware Card. The concept of a RAM version of Integer Basic was first implemented by James McBride when he successfully relocated it on a friends Apple II + with 32K RAM. It worked great, with the exception that all programs had to be SAVEd to or LOADed from cassette tape.

A friend of McBrides, Garth Hitchens, was asked to work out the required DOS modifications to enable programs to be LOADed and SAVEd on disk. The next step was to cause the RAM Integer to emulate the protocols of RAM Applesoft, and this was soon accomplished.

At this point RAM Integer Basic could do almost anything the ROM version could do except for certain programs which CALL on routines in the Integer ROM which are not actually a part of the Integer Basic language, and, in addition, the RAM version was not relocatable. After more effort, a relocatable version was developed, followed shortly by a self-relocating version. relocating version.

RAM Integer Basic had become a reality. Now, in discussion with the [illegible] ectors of A.P.P.L.E. in connection with making this available to the member ship, it was decided to enhance the language beyond its original power. First the unused RNDX token was modified to USER and 20 functions such as CHRS, FLASH and Output Hex were predefined, with another 6 functions available for definition by the user. And then a DOS ONERR GOTO routine was added in to complete the package.

McBride and Hitchens are both in their mid-teens and have more than proven their capabilities as programmers. Their combined teamwork has resulted in the production of another product that may be enjoyed by all A.P.P.L.E. members.

## GETTING STARTED WITH THE INTEGER BASIC + SYSTEM

The Integer Basic Plus System has been designed for ease of use. Wherever possible it duplicates exactly the functions of the standard ROM based Integer Basic. It is compatible with the Program Line Editor if PLE is first RUN from Applesoft before entering Integer.

If this is your first experience with Integer Basic, you should buy and study the Integer Basic Programming Manual, published by Apple Computer, Inc. and available from your local dealer.

To start the system up, just insert the Integer Basic Plus System master diskette in your disk drive and follow your normal booting procedure, which is described in the DOS 3.2 manual. For a regular Apple II+, it is a matter of turning the power off, inserting the diskette and turning the power back on, which will result in an autoboot.

After the disk has booted, you will be greeted first by the "Hello" program and the Applesoft prompt ( ] ). This is the time to RUN Program Line Editor if you care to.

Also at this time, if you already have Integer Basic in ROM, either on the main board of an Apple II or on a firmware card on an Apple II +, and you wish to use Integer Basic Plus rather than the ROM version, enter BRUN ROM INTEGER. This will confuse DOS to the extent that it will not know you have Integer ROM on board and will allow you to use the RAM version instead.

To Enter Integer Basic Plus, just type "INT" followed by a carriage return and Integer Plus will load, and give you the standard Integer Basic prompt (>), indicating that you now may proceed to LOAD and SAVE programs in the normal manner and use Integer as needed.

*CAUTION*: You may use the FP command to return to Applesoft, but any attempt to LOAD or RUN an Integer program from Applesoft without first entering an "INT" will not be successful. This may be modified in a later version.

## TRANSFERRING INTEGER TO OTHER DISKETTES

The DOS contained on your Integer Basic Plus System master diskette is a unique, highly modified 3.2.1 DOS. It is recommended that you immediately make a back up copy of the System master in the event of its subsequent failure. A copy may be made using the Apple copy program or the Single/Dual Drive Copy program from our diskpak 6. In each case, just follow your usual copy procedures.

In some cases, it may be desirable to transfer the Integer program to another previously initialized diskette. If this is done, it is imperative that the "Update 3.2.1" program included on your Integer Basic Plus System master diskette be used. To do this, type the following commands from Applesoft mode:

```
] LOAD INTEGER          (on Integer master diskette)
] SAVE INTEGER          (on new diskette)
] BRUN UPDATE 3.2.1     (on Integer master diskette)
```

When the Update 3.2.1 program BRUNs, you will be prompted what to do. Just type the file name of the hello program you have assigned to the new diskette when requested. After the new diskette has been updated, you may use it just as you previously used the master.

Note that *ONLY* the Update 3.2.1 program from the Integer Basic Plus System master diskette may be used. The same program from the Apple master diskette will *NOT* work because the Apple diskette has only standard 3.2.1. DOS.

## USER (x) FUNCTIONS

USER (x) functions are a method of expanding standard Integer Basic with additional commands. There are 20 pre-defined USER(x) commands and an additional 6 may be defined by the user. The USER(x) commands may be entered into a program in one of two ways:

1. `100 PRINT USER(x);` (where x represents a number from 0 to 25.)
2. `100 N = USER(x)` (where x represents a number from 0 to 25.)

Method 1 is preferable because it is processed more rapidly than method 2. The USER(x) function may be incorporated into a PRINT statement, using semicolons, etc., just as you would any standard PRINT function. An example of printing the word "HELLO" to the screen in INVERSE video would be as follows:

```
110 PRINT USER(1); "HELLO"; USER(2) : END
```

This would change the output mode to INVERSE, print "HELLO" and then restore the output mode to NORMAL. Note that the PRINT command need be used only once.

USER(x) functions 16 and 17, which return a HEX$ or CHR$ respectively may be used as in the following example

```
120 POKE 1,255: PRINT USER(16): END
```

This would return "FF" to your display. A more detailed explanation appears in the following list of USER(x) commands and their application.

## TABLE OF USER (x) FUNCTIONS

0 CLEAR TEXT SCREEN. This will clear the text screen to the current output mode. If the current mode is INVERSE, then PRINT USER(0) would return an all white screen area.

1. INVERSE. Sets the INVERSE mode; all text output to the screen will now appear as black characters on white.
2. NORMAL. Returns the video display to the NORMAL (white on black) mode.
3. FLASH. Sets character output to alternate between NORMAL and INVERSE. (See 1 and 2, above).
4. CURSOR UP. Moves the cursor up one space. Enter the following line to test this function: 130 CALL -936: VTAB16: TAB2: PRINT "HELLO"; USER(4); USER(4);: END This function supplements VTAB.
5. CURSOR DOWN. Moves cursor down one space. Functions as 4, above.
6. CURSOR LEFT. Moves cursor one space to the left.
7. CURSOR RIGHT. Moves cursor one space to the right.
8. FAST MIXED GR CLEAR. Clears the screen to mixed graphics mode (four lines of text on the bottom) to the color previously set with the COLOR = command.
9. FAST FULL GR CLEAR. Clears the screen to full graphics mode (no text panel) to the color previously set with the COLOR = command.
10. CLEAR TO EOP. Clears the text screen from the present cursor position to the end of the page.
11. CLEAR TO EOL. Clears the text screen from the present cursor position to the end of the current line.
12. SCROLL UP. Scrolls the text screen up one line, filling in at the bottom with a blank line of text. If the video mode were set to INVERSE, the fill line would appear as white. If the video mode was set to NORMAL, there would be no apparent difference. Experimentation with the four SCROLL commands, USER(12) to USER(15) inclusive, will show you how to use these functions to draw INVERSE or FLASHing borders, etc., on your text screen.
13. SCROLL DOWN. Functions exactly as 12, above, except scrolls down with top fill.
14. SCROLL LEFT. Functions exactly as 12, above, except scrolls left with fill on the right.
15. SCROLL RIGHT. Functions exactly as 12, above, except scrolls right with fill on the left.

16. HEX$. Returns a hexadecimal value equivalent to the decimal value previously POKEd into location 1, in the range 0 to 255. Example:

    POKE 1,128: PRINT USER(16) would return a hex value of 80.

17. CHR$. Returns to the screen the ASCII character whose value has previously been POKEd into 1. Example. POKE 1,223. PRINT USER(17) would print an underscore "_" to the screen or, POKE 1,225: PRINT USER(17) would print lower case "a" to the screen if you have the Dan Paymar chip and can display lower case. (Else it would print a "!"). Note that unlike Applesoft, the Integer Basic ASCII character set of NORMAL characters is in the range 128 to 255. POKEing 1 with a value less than 128 will result in outputting either FLASH or INVERSE characters.
18. BASIC ADDRESS. Stores the high byte of the beginning of Integer Basic Plus in location $00. This is used internally by Integer Basic Plus, but may prove helpful in patching in routines that call on the Integer ROM.
19. DOS ONERR GOTO. This routine, based on one written by Andy Hertzfeld, is a DOS error handling routine that functions in a manner similar to the Applesoft ONERR GOTO. Its purpose is to prevent a disk-based program from aborting when an error is encountered, by instructing the program to jump to a specified line number.

    It is enabled by entering a "PRINT USER(19)", POKEing the error handling line number, MOD256 into 10, and POKEing the error handling line number /256 into location 11. When an error occurs, the error type will be found by PEEKing at 7 and the line number of the line that caused the error will be returned in locations 8 and 9. Standard DOS error codes, as shown on Pages 114-115 of the DOS 3.2 manual are used. The following is a short program to demonstrate the use of DOS ONERR GOTO

```
10   REM DOS ONERR GOTO DEMO
20   PRINT USER(19);
30   POKE 10, 1000 MOD 256: POKE 11, 1000/256
40   D$ = "": REM CTRL D IN QUOTES
50   PRINT D$; "CATALOG"
60   GOTO 50
70   REM OPEN DRIVE DOOR TO GENERATE
          A DISK I/O ERROR

1000 REM ERROR HANDLING ROUTINE
1010 PRINT "ERROR TYPE : "; PEEK(7)" ";
1020 PRINT "AT LINE : "; PEEK(8) + PEEK(9) * 256
1030 REM CLOSE YOUR DRIVE DOOR
1040 PRINT: GOTO 50
```

*NOTE: Use of the DOS ONERR GOTO routine does not inhibit the printing of DOS error messages, which can be a DANGEROUS practice that in direct mode could conceivably lead to damage of your diskette or program, however, for those that care to, the following two program lines will disable and re-enable the DOS error message. Both lines must be used within the program, with 32767 just preceding or a part of the END statement.*

```
32766  ERRMSG=256*( PEEK ( 978 )-256
       *( PEEK ( 978 )>127 ) )+2524: POKE
       ERRMSG,18: REM DISABLE ERRMSG

32767  POKE ERRMSG,4: END : REM RE-ENA
       BLE ERRMSG
```

20 to 25 inc. Are undefined and may be utilized by users for their own purposes. See the following section to define your own functions.

## DEFINING YOUR OWN FUNCTIONS

In addition to USER(0) through USER(19) which are predefined, you may set up your own functions with USER(20) through USER(25). The user defined functions will allow you to enter jumps to any memory location, indexed by its address in the variable RTN. For the sake of consistency, user defined functions should be defined within the program that utilizes them. A text file, "USER DEFINE" has been established for this purpose.

If you wish to add a function to an existing program, just LOAD the program and EXEC USER DEFINE, set the variable FUN to equal the number to be assigned to the function and set RTN to equal the address of the routine to be called by the function. In the following program example, RTN = -936. As a last step, enter a line in your program to GOSUB 32760.

```
> EXEC USER DEFINE
10    REM PROGRAM TO DEFINE AND DEMO
            USER(20)
20    FUN = 20: RTN = -936
30    REM VARIABLES ARE NOW SET UP
40    GOSUB 32760: REM DEFINE THE FUNCTION
50    REM USER(20) NOW DOES A CALL -936
60    REM NOW LETS TRY IT

100   PRINT USER(20), "THE SCREEN IS
            NOW CLEARED": END
32760 - 32765 (User define routine)
```

In addition to defining the unused USER(x) functions, you may also redefine any of the existing ones to suit your own needs, e.g., USER(3) could be redefined to clear the screen, even though it was originally defined to set the FLASH mode. Note that these changes will not be present when the system is rebooted, but they will be carried from program to program.

## STOP LIST FEATURE

Integer Basic Plus also includes a stop list feature. To temporarily halt a program listing, enter a CTRL S. Any other key will resume the listing. A CTRL C will abort the listing altogether and return control to Integer Basic.

## PROGRAM COMPATIBILITY

**Integer Basic Plus to standard Integer Basic**

Programs written in Integer Basic Plus will run on a standard Apple II provided that the USER(x) function is not used and provided that there are no CALLs to special Integer Basic Plus locations.

**Integer Basic Plus to Integer Basic Plus**

Programs written in Integer Basic Plus utilizing CALLs to special Integer Basic Plus locations will run only on a system with the same memory size as yours. Programs written using only the USER(x) function will run on any Integer Basic Plus system.

**Standard Integer Basic to Integer Basic Plus**

Any standard Integer Basic program should run under Integer Basic Plus with the exceptions of programs that reference addresses in the Integer ROM. However, these programs may be modified easily so that they will run properly. A program named "CONVERT" has been included to change Integer ROM addresses in any binary program, and some Integer programs where the address is in Hex.

To use this feature, boot the Integer Basic Plus System master diskette and RUN Convert. Place the diskette with the program to be converted in the disk drive and respond with the name of the program to be converted when prompted to do so. The process is automatic, and the converted program will be SAVEd to disk with the name of the original program followed by the suffix ".CON".

Convert will convert most Assembly Language programs but will convert only those Integer Basic programs in which the ROM address is given in hexadecimal, as in a program using Lam's Monitor String Routine.

*Users are cautioned again that any programs utilizing either special addresses or the USER(x) function will* **NOT** *run on standard Integer Basic. Therefore these features should not be used in programs for commercial use or for others who do not have Integer Basic Plus available.*

# QUICK REFERENCE LIST - TED II+
# COMMANDS AND PSEUDO-OPS

## Executive Mode

| | |
|---|---|
| # | SET DRIVE # |
| C | CATALOG |
| D | DISK COMMAND |
| M | MONITOR COMMAND |
| L | LOAD SOURCE FILE |
| S | SAVE SOURCE FILE |
| O | OBJECT CODE SAVE |
| A | APPEND FILES |
| T | TED (ENTER ED/ASM) |
| Q | QUIT |

## Edit Module

### System Commands

| | |
|---|---|
| NEW | Clear existing source file |
| LO*mem:* | Set lower limit of source |
| HI*mem:* | Set high limit of source |
| PR# | Same as BASIC PR# |
| LEN | Display source length |
| LOAD | Load cassette source |
| SAVE | Save cassette source |
| TABS | Set or clear tab stops |
| *List* | List lines as specified |
| *Find* | Find line # or string |
| ASM | Assemble source file |
| *Quit* | Quit TED and enter EXEC |

### Entry Commands

| | |
|---|---|
| *Add* | Add lines mode |
| *Insert* | Insert lines mode |
| *Delete* | Delete lines mode |
| COPY | Copy one range to another |
| *Change* | Change string to another |
| *Edit* | Edit line #, range or string |

## Edit Mode Sub-Commands

| | |
|---|---|
| Carriage rtn or Ctrl M | Accept entire line |
| Ctrl Q | Accept to cursor |
| Ctrl F | Find character |
| Ctrl I | Insert character |
| Ctrl O | Insert ctrl char |
| Ctrl D | Delete character |
| Ctrl R | Restart Edit |
| Ctrl C | Exit edit mode |

## TED System Pseudo ops

| | |
|---|---|
| OBJ | Set OBJ address |
| ORG | Set ORG address |
| EQU | Assign address to label |
| AST | Print asterisks |
| LST ON | Turn on listing |
| LST OFF | Turn off listing |
| PAG | Clear screen |
| PR # | Output to PR# |
| SYM | Print symbol table |
| END | End of program |

## Optional 6502 Instructions

| | |
|---|---|
| BLT | Same as BCC |
| BGE | Same as BCS |

## Functional Pseudo-ops

| | |
|---|---|
| ASC | Enter ASCII code |
| DCI | Same except last byte |
| HEX | Enter hex data |
| DW | Enter one byte of hex |
| DS | Reserve bytes |
| DA | Write label address |
| DB | Write label adr low |